Experiments in Amorphous Geometry

Ellie D'Hondt[1] and Theo D'Hondt[2]

[1] Foundations of Exact Sciences (FUND)
[2] Programming Technology Laboratory
Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium
{eldhondt|tjdhondt}@vub.ac.be
tel +322629 {3474|3480}
fax +322629 {3495|3525}

Abstract. Amorphous computing is a recently introduced paradigm that favours geometrical configurations. The physical layout of an amorphous computer is based on a possibly irregular and error-prone planar distribution of a large number of simple processing components; this is well-suited for handling spatial structures. It has come to our attention that the well-known and long-since established discipline of computational geometry could benefit from the amorphous computing approach. It seemed natural to refer to this approach by the notion of amorphous geometry. Although at this stage our exploration of this concept is fairly modest, we feel that our experiments are sufficiently convincing and merit further study.
Exploring the possibilities of amorphous geometry should show that amorphous computing can deal with various classes of problems from computational geometry and may eventually expand the field considerably while providing a basis for useful applications. Especially our preliminary results concerning the triangulation problem are of importance since it is very representative case in computational geometry: it covers a whole family of problems.

Keywords. amorphous computing, computational geometry, triangulation

1 Introduction

The context in which this paper is set is a newly emerging domain in computing called *Amorphous Computing* [1]. It was developed in anticipation of the rapidly developing fields of microfabrication and cellular engineering. These support the mass production of small computational units with limited power, provided that there need be no guarantee that each unit works faultlessly. A random distribution of such particles, equiped with a local communication protocol, constitutes the basic model of an amorphous computer. However, while producing a system composed of such units is within our reach, there as yet exist no programming systems applicable to it. Amorphous computing is precisely the paradigm trying to fill the gap between the construction and the programming techniques required for an amorphous computer. More concretely, amorphous computing addresses the important task of identifying the appropriate organising principles and programming methodologies for obtaining predefined global behaviour through local interaction only; individual amorphous computing particles cannot rely on any global knowledge about the system they are a part of. Since disciplines such as biology and physics often operate within this context, they were adopted as a source for metaphors applicable within the field of amorphous computing, which among others led to the development of the *Growing Point Language* (GPL) [2]. For a more detailed overview of amorphous computing and a comprehensive list of references we refer to [4].

We propose introducing amorphous computing into the venerable field of *Computational Geometry* (see for instance the excellent [3]). Computational geometry serves as a foundation for various disciplines, including but not limited to operations research, artificial intelligence and robotics. Nearly as old as computing itself, it encompasses a vast body of knowledge rooted in mathematics, algorithms and programming. The fact that amorphous computing addresses problems that exist in physical−generally planar−space, led us to the conviction that it might be a useful addition to the domain of computational geometry. The physical layout of an amorphous computer serves as a medium for the representation of geometrical configurations; therefore it seems natural to introduce the notion of *amorphous geometry.* Moreover, we can imagine a complete range of useful applications, such as smart surfaces interacting with their environment in any possible way. Thinking about amorphous computing can be done in terms of representative applications; one of them is an active anechoic wall that reduces noise by using an audio-sensing network of amorphous computing particles dissolved in the paint covering the wall. We like to fix our ideas about amorphous geometry by thinking about examples such as active hospital floors that can compute and visualise paths to be followed by people walking on them.

Exploring the possibilities of amorphous geometry is an immense task. In order to show that amorphous computing can deal with more than a few problems taken from computational geometry, much more needs to be done than we can present here. We will limit ourselves to a number of modest experiments that nevertheless illustrate the point we want to make; all experiments are presented within the context of GPL. Section 2 handles the amorphous construction of polygons. Additionally, and in order to satisfy the need for more expressiveness, several extensions to GPL and their implementations are presented. Next, section 3 covers diagonal construction, as well as its application to the triangulation problem. We conclude in section 4 with some current and future work.

2 Polygon Construction

The problem of drawing a polygon can be divided into the subproblems of constructing line segments from one vertex to another. In order to do this, one endpoint has to initiate a "search" for the other endpoint through local communication only, without knowing the global direction in which the latter is to be found. Concretely, the representation of an edge in GPL is as follows:

```
(define-growing-point (edge next-vertex-pheromone next-vertex-material)
  (material edge-material)
  (size 0)
  (tropism (ortho+ next-vertex-pheromone))
  (actions
  (when ((sensing? '(: next-vertex-material))
    (terminate))
    (default
    (propagate next-vertex-pheromone)))))
```

The principal concept in GPL is the *growing point*, a locus of activity that describes a path through connected elements in the system. At all times, a growing point resides at a single location in the GPL domain, called its *active site*; for example, the initial active site for the above growing point is one of its endpoints. A growing point propagates itself by transferring its activity from a particle to one of its neighbours according to its *tropism.* At its active site, a growing point may deposit *material* and it may secrete *pheromones.* A growing point's tropism is specified in terms of the neighbouring pheromone concentrations; in this case, `ortho+` directs the growing point towards increasing concentrations of `next-vertex-pheromone`. As an edge is generated, the generator process itself will require some cooperation from the second endpoint in order to produce the desired line segment. This is indicated by the presence of `next-vertex-pheromone` and `next-vertex-material` in the code above. Hence, a growing point is installed at the second endpoint in the following way

```
(define-growing-point (vertex vertex-pheromone vertex-material)
  (material '(: vertex-material))
  (size 0)
  (actions
  (secrete+ EDGE-LENGTH vertex-pheromone)))
```

Note that the above growing point definitions take a pheromone and a material parameter. The latter is used in the `sensing?` and in the `material` expression, using the double point GPL syntax for parameterised materials. Pheromone parametrisation however, is not present in standard GPL. We included it in its framework since we felt GPL would gain considerably in expressiveness by allowing pheromones as parameters. Indeed, without this feature each edge of a polygon to be constructed would require a different growing point definition in order to avoid pheromone interference, involving a lot of duplicate code. In order to construct a quadrangle, for example, we would have to use a unique pheromone and material for each vertex, and define an associated edge-like growing point that grows towards that particular pheromone. With pheromone parametrisation, a quadrangle is constructed using only the above two growing points in the following way

```
(with-locations (a b c d)
 (at a (start-gp (vertex 'a-pheromone 'a-material))
   (--> (start-gp (edge 'b-pheromone 'b-material))
    (--> (start-gp (edge 'c-pheromone 'c-material))
     (--> (start-gp (edge 'd-pheromone 'd-material))
      (start-gp (edge 'a-pheromone 'a-material))))))
 (at b (start-gp (vertex 'b-pheromone 'b-material)))
 (at c (start-gp (vertex 'c-pheromone 'c-material)))
 (at d (start-gp (vertex 'd-pheromone 'd-material))))
```

Here `-->` is the `connect`-command, which allows us to connect several growing points. Note that in the above, explicit pheromone and material names are required to be quoted, which constitutes a change in syntax with respect to the previous version of GPL.

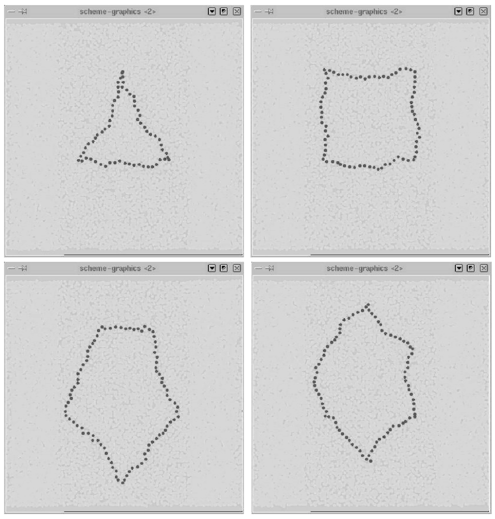

Fig. 1. Several polygons constructed with GPL.

Simulation results for the construction of a quadrangle with the GPL-illustrator in the above way, as well as for the analogous construction of a triangle, pentagon and hexagon are shown in figure 1 (note that the segments only approximate a straight line). It may be inferred from these results that arbitrary polygons can be constructed in an amorphous way.

3 Diagonal and Triangulation

Let us now look at the problem of constructing a convex polygon's diagonals, defining a diagonal as a straight line between non-adjacent vertices that lies entirely on the inside of the quadrangle. One extra complication in this case is that in order for GPL to know what part of the domain is the polygon's inside, we need to establish an additional growing point at an arbitrary internal point of the polygon[1] – referred to as `inside` below. Its only task is the secretion of `center-pheromone`, which directs the diagonals towards the inside of the quadrangle. However, in order for the diagonal to reach an opposite vertex, at one point the positive orthotropism towards `center-pheromone` has to be eliminated. Therefore, we have to split up the construction of the diagonal as shown if figure 2.

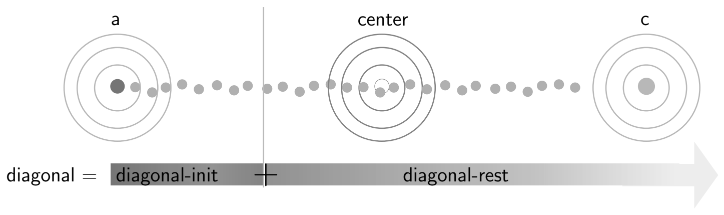

Fig. 2. Composition of the `diagonal` growing point.

More concretely, in case of a quadrangle the `diagonal` growing point is a combination of two growing points: `diagonal-init`, which ensures thatthe diagonal starts off towards the inside of the quadrangle, and `diagonal-rest`, which constructs the rest of the diagonal. The growing point definitions required for the above scheme are as follows – note that we included pheromone parametrisation in view of having to construct several diagonals at the same time.

```
(define-growing-point (diagonal-init length)
   (material diagonal-material)
   (size 0)
   (tropism (ortho+ center-pheromone))
   (actions
     (secrete+ 1 diagonal-pheromone)
     (when ((< length 1)
            (terminate))
            (default
            (propagate (- length 1))))))

(define-growing-point (diagonal-rest endpoint-pheromone endpoint-material)
     (material diagonal-material)
     (size 0)
     (tropism (and (ortho- diagonal-pheromone)
                   (ortho+ endpoint-pheromone)))
     (actions
       (secrete+ 1 diagonal-pheromone)
       (when ((sensing? endpoint-material))
```

[1] The choice of an internal point is inspired by [2]; alternatives could have been: considering polygons filled with a specific material, or two-material edges.

```
              (terminate))
             (default
              (propagate)))))

(define-growing-point (diagonal endpoint-pheromone endpoint-material)
     (material diagonal-material)
     (size 0)
     (actions
       (--> (start-gp diagonal-init 2)
            (start-gp diagonal-rest endpoint-pheromone endpoint-material))))
```

As one can see, `diagonal-init` determines the initial direction, while `diagonal- rest` behaves like a growing point with inertia, maintaining the same direction by growing away from its own pheromone. In case of general polygons, one should define `diagonal-rest` as being positively orthotropic towards all possible endpoints; for a pentangle for example, this requires `diagonal-rest` to have two different pheromone parameters, one for each possible diagonal endpoint.

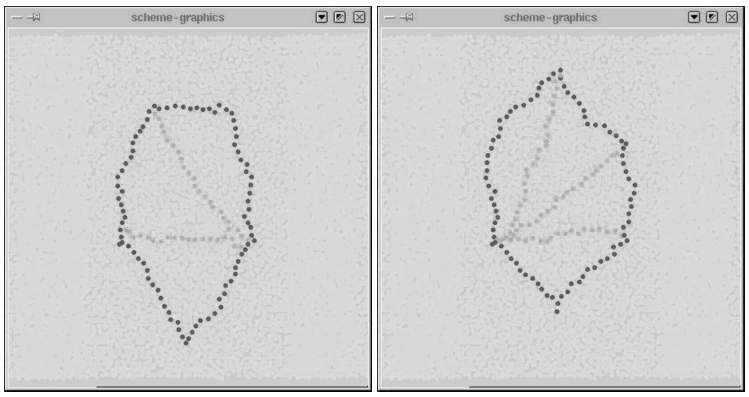

Fig. 3. Triangulating a polygon.

Once diagonal construction is possible, we can envisage tackling the triangulation problem by allowing diagonals to grow from each vertex. Triangulating a polygon is an very representative case in computational geometry: it covers a whole family of problems. In figure 3 we show some initial results from an experiment to apply amorphous geometry to the triangulation of a polygon. Again, pheromone parametrisation proved valuable, since otherwise we would have had to define a different growing point for each constructed diagonal. However, in order to solve this problem in general still requires research. This is due to the following: an ideal solution would be to allow diagonals to grow from each vertex with the additional constraint that they cannot intersect; once a triangulation is found, the construction of superfluous diagonals should be cancelled. At this stage however, it is not entirely clear to us how to cancel growing points while they are still being constructed. A possible solution is to first search the opposite vertex, then reconstructing the found diagonal. In this case, we could only display the diagonal material when backtracking, thus avoiding a cluttered result due to unfinished growing points desperately trying to find a vertex while none are left. However, further investigations are required in order to clarify this issue. As for non-intersecting diagonals, these can be implemented easily by including the clause (avoids diagonal-pheromone) in a diagonal's definition. In spite of this, we found that for the order of simulations we could run the `avoids` clauses resulted in stuck growing points. This is because for not too large simulations the density of particles is too small to leave enough choice for a diagonal growing point whilst encountering other diagonals. Therefore, we only established as many diagonals as required for the triangulation in question, thus eliminating the need for the `avoids`-clause as well as that of cancelling superfluous diagonals once a triangulation has been constructed.

It should be clear from the above reasoning that solving the triangulation problem is entirely within reach of GPL. The problem of shutting down eager growing points after a solution is found is practical rather than theoretical, since the main goal of constructing a polygon triangulation is reached anyhow.

4 Conclusion and Future Work

Amorphous computing is an interesting new programming paradigm that is based on the notion that a large network of computational particles can be embedded in physical surfaces. It seems almost obvious to propose it as an alternative to some of the more conventional computational techniques used to tackle geometrical problems. In this paper, we have suggested the notion of amorphous geometry to explore this idea. Typically, each amorphous processing particle governs a geometrical patch, such that the whole medium cooperates in finding a global solution to a particular problem at hand. We only scratched the surface by considering points, lines, polygons and diagonals; we showed that they can be simulated by very simple programs, composed in the growing point language. We rapidly discovered that the expressiveness of GPL is too limited for tasks as elementary as generating the edges of a polygon from the vertices. However, introducing a particular kind of parametrisation in GPL seems to solve the shortcoming, and did not pose any undue problem.

Extending GPL into a sufficiently expressive language so as to cover typical problems from computational geometry would seem to be a worthwhile continuation of the work described in this paper. In particular, solving the triangulation problem in general is an interesting challenge. We expect it to be possible to continue from this basis on towards a whole family of related geometrical problems, such as, for example, traject planning for robots.

References

1. H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman, and R. Weiss: *Amorphous computing.* Communications of the ACM, 43(5), May 2000.
2. D. Coore: *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computing.* PhD thesis, MIT, 1999.
3. M. de Berg and M. van Kreveld and M. Overmars and O. Schwarzkopf: *Computational Geometry: Algorithms and Applications.* Springer (1998) second edition
4. E. D'Hondt: *Amorphous Computing.* MSc thesis in Computer Science, Vrije Universiteit Brussel (2000); available at `http://student.vub.ac.be/~eldhondt/PDF directory/thesisAmorphous.pdf`